7-1-1988

Methods used in the development of computer programs capable of boolean operators on polyhedrons

Timothy W. Flood
Pittsburg State University

METHODS USED IN THE DEVELOPMENT OF COMPUTER PROGRAMS CAPABLE OF BOOLEAN OPERATIONS ON POLYHEDRONS

A Problem Submitted to the Graduate Division in Partial

Fulfillment of the Requirements for the

Degree of Master of Science

By

Timothy W. Flood

PITTSBURG STATE UNIVERSITY

Pittsburg, Kansas

July, 1988

ACKNOWLEDGMENTS

The author of this paper would like to thank Dr. Gary L. McGrath for his constant help and encouragement, not only on this paper, but also during my five years here at Pittsburg State University. I would also like to thank the rest of the people in the Math Department for their support.

ABSTRACT

This paper discusses a method which can be employed on a computer to allow the computer to perform Boolean operations on polygons and polyhedrons. Although current literature is full of algorithms which are vital in the construction of such a program, not many complete algorithms are available. The method described here employs several of the current algorithms and joins them together with other information to produce a complete package. The objects are stored using a boundary representation in linked lists. The polyhedrons are represented by the polygons that compose their faces. The polygons are processed by intersecting each line segment of a given polygon with all of the line segments the other polygon. The new line segments, induced by these intersections, are introduced and the fundamental cycles of the graph evaluated as to the region they bound and selected accordingly.

LIST OF TABLES

TABLE PAGE

I. Classification of Cycles by Content 22

II. Selection Rules for Boolean Operations . . . 24

III. Characterization of Facets by Adjoining Regions 31

LIST OF FIGURES

FIGURE PAGE

1. Polygon 1 3
2. Polygon 2 4
3. Polygon 1 with the Vertices and Edges Numbered in the Order of Traversal 6
4. Polygon 2 with the Vertices and Edges Numbered in the Order of Traversal 7
5. Polygons 1 and 2 9
6. Numbering of Half Edges and Vertices 11
7. Numbering of Opposite Half Edges 12
8. Building of the First Three Cycles 15
9. Remaining Cycles 16
10. Construction of P in the Interior of Cycle 3 17
11. Point P in the Interior of Polygon 1 19
12. Examples of the Difficulties with the Cycle Classification Method 21
13. Polygon 1 Intersect the Complement of Polygon 2 23
14. Polygon 1 Union Polygon 2 25
15. Polyhedrons 1 and 2 27
16. Numbering of Facets in Polyhedron 1 29
17. Numbering of Facets in Polyhedron 2 30

METHODS USED IN THE DEVELOPMENT OF COMPUTER PROGRAMS CAPABLE OF BOOLEAN OPERATIONS ON POLYHEDRONS

The development of computer programs capable of polygon and polyhedron Boolean operations is a rapidly growing field of interest. It is of interest in computer science, especially in three-dimensional computer graphics, for determining the visible parts of solid objects. It is also of interest in the design of automated processes, in order to determine if the parts of the machine will collide or not.

One of the first considerations in developing this type of program is what type of data structure is best suited for the application. Three-dimensional polyhedrons can be represented as a finite number of two-dimensional polygons describing the faces of the three-dimensional polyhedron, so the representation of two-dimensional polygons will be discussed first. There are several data structures currently being used. One structure, that is memory efficient, stores the vertices in a linked list, in the order that they occur in the polygon. Another structure, which is more conducive to computer graphics but uses more memory, is to break the polygon into a finite number of line segments, where the beginning and ending vertices of the line segments are stored in linked list in the order they occur as the polygon is traversed in a predetermined direction. Since the major emphasis of this paper is computer graphics, the latter method will be used throughout.

Consider, for example, polygon 1 shown in figure 1. First a starting vertex and an orientation must be chosen. A positive orientation, in which the polygonal region is to the left as the perimeter is traversed in a counter-clockwise direction will be used throughout this paper. In polygon 1, vertex A is arbitrarily chosen as the starting vertex. The polygon is stored as directed line segment, or half edge, AD, followed by DH, then HE, EF, FG, GI, IJ, JC, CB, and finally BA completing the cycle. As stated earlier, this method uses considerable memory, but each individual line segment can be easily plotted and thus the entire polygon plotted. This method gives a well defined polygonal region, provided the polygonal boundary is not self-intersecting, hence self-intersecting polygons will not be considered. This method works for polygonal regions with holes, consider polygon 2 in figure 2. It is represented as two cycles, one cycle being KQ, QR, RL, and LK; and the other cycle MN, NP, PO, and OM.

Figure 1

Polygon 1

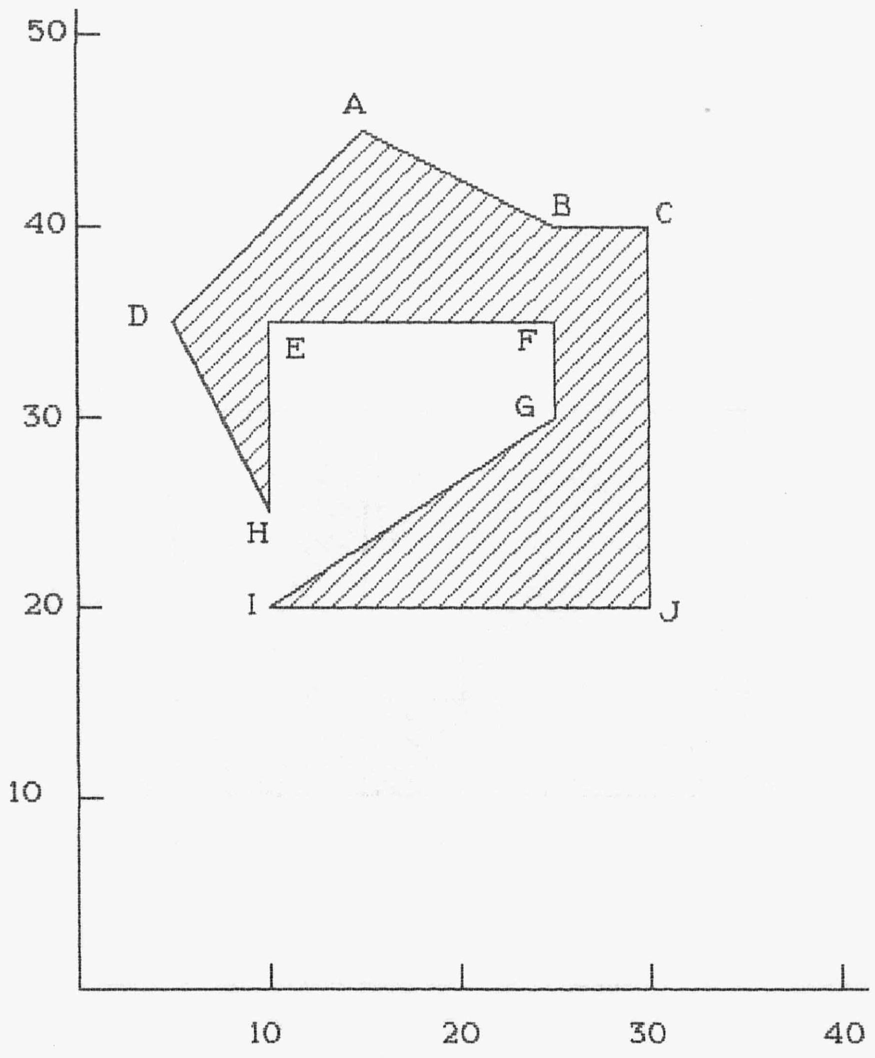
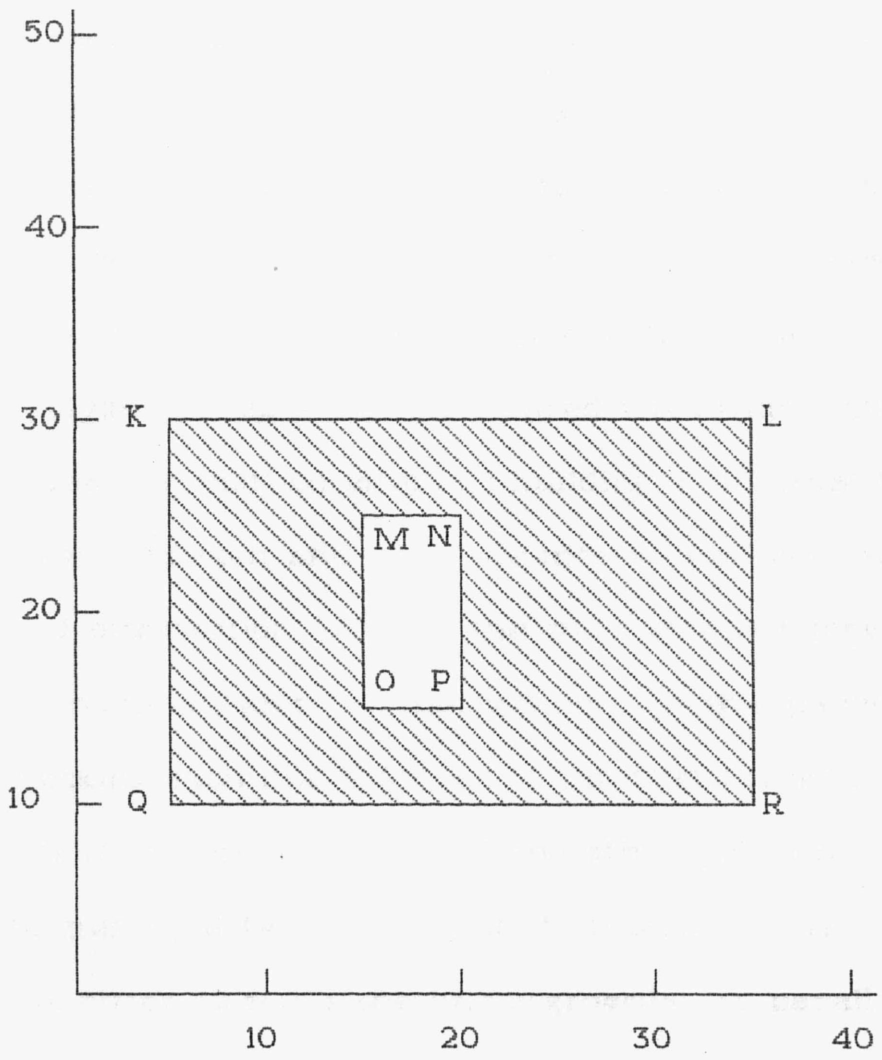


Figure 2

Polygon 2



Once polygons can be stored in a manner in which they can be operated on, one can begin to process the data. The vertices and half edges are numbered in the order in which they are traversed, as in figures 3 and 4. Several methods are currently being used to determine the intersection of the polygons. One method involves building a grid over the polygonal regions and checking which line segments occur in each of the cells. Then only those line segments which occur in the same cell need to be evaluated to see if they intersect. This method requires a considerable amount of preprocessing and additional memory, and would only be advantageous when one polygon is repeatedly compared with many other polygons. Another method employs building a minimal box containing each polygon and then the intersection is evaluated in the box of intersection. One of the most straight forward, easiest to implement, and the method used in this paper involves checking each line segment of the first polygon against each of the line segments of the other polygon.

In determining if two line segments intersect it is necessary to check to see if the line segments are parallel, or if they are collinear. If the cross product of vectors lying on these line segments is zero, then the line segments are parallel. If the vectors are found to be linearly dependent,

Figure 3

Polygon 1 with the Vertices and Edges

Numbered in the Order of Traversal

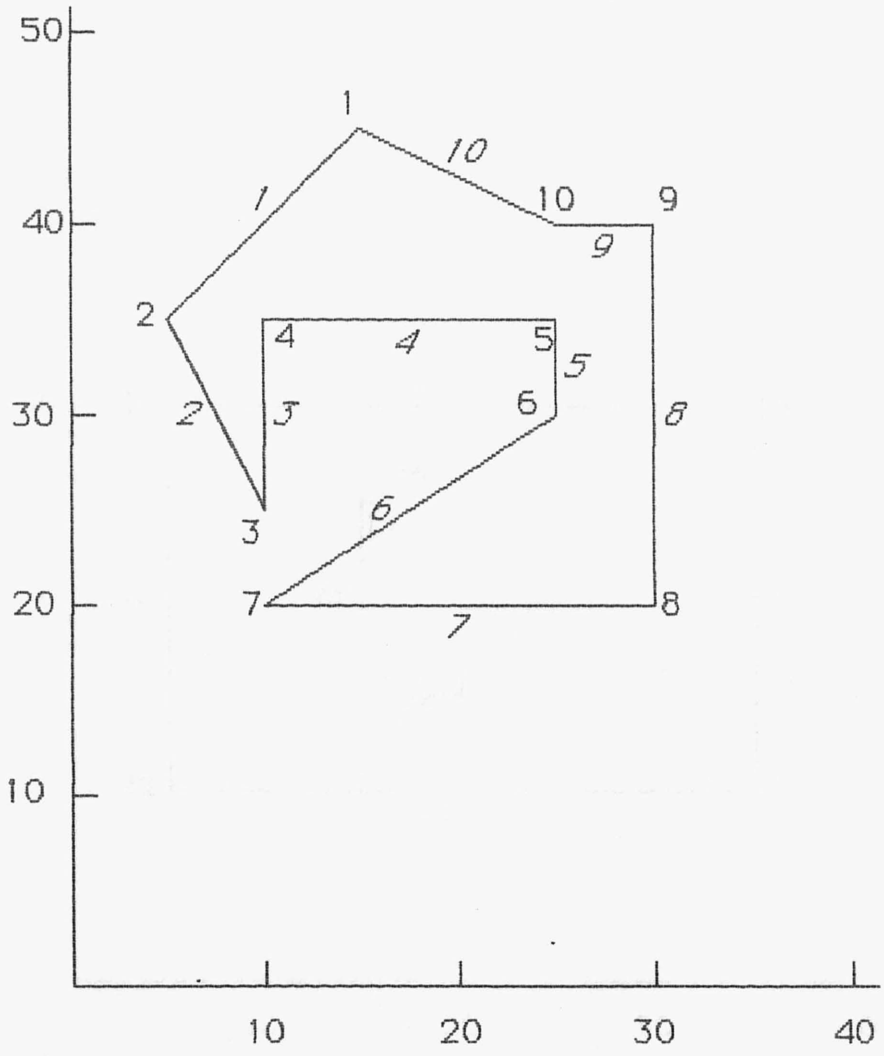
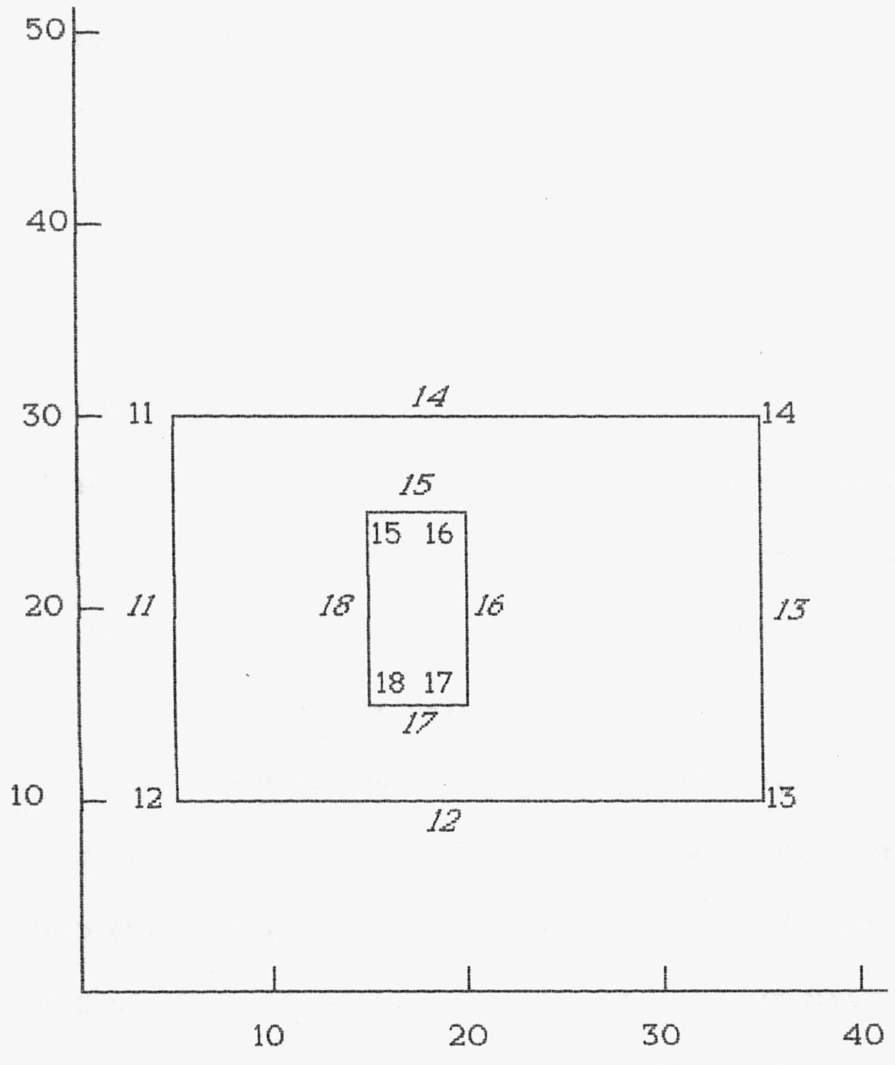


Figure 4

Polygon 2 with the Vertices and Edges Numbered in the Order of Traversal

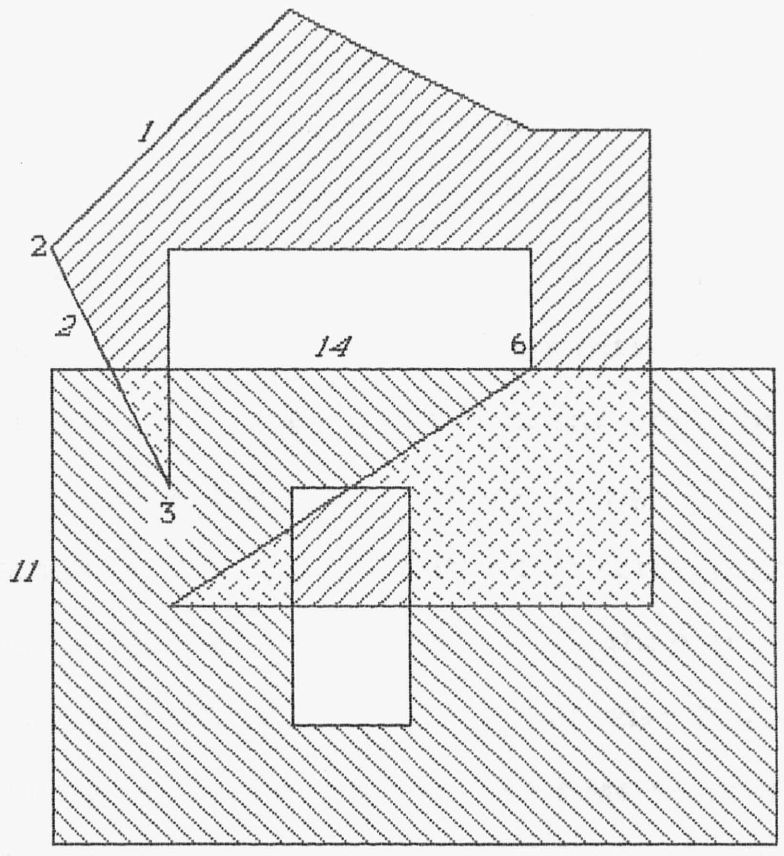


the line segments are collinear and the end points of each are checked to see if they are on the other line segment and the intersection thus determined. If the line segments are not collinear or parallel then the lines through the end points must intersect somewhere. The point of intersection can be determined by solving the system of equations of the lines simultaneously. Once the point of intersection is found, it must be determined if the point is on both line segments. If it is on both line segments, then the line segments do intersect.

Figure 5 shows the two polygons in the same plane. The points of intersection must be determined. The line containing line segment 1 would be found to intersect several of the lines defined by the line segments of polygon 2, such as line segment 11, but the point of intersection does not occur on line segment 11 so the line segments do not intersect. Line segment 2 is found to intersect line segment 14, and the point of intersection is found to be (7.5,30). This new vertex must be linked into the polygons, between vertices 2 and 3 in polygon 1 and vertices 14 and 11 in polygon 2. This new vertex also divides the two old line segments, creating four new line segments. This process is continued until all of the line segments in polygon 1 are compared to each line segment

Figure 5

Polygons 1 and 2



in polygon 2. A unique case occurs where line segment 14 intersects vertex 6. This does not introduce a new vertex but it does divide line segment 14 into two new line segments. Once all of the points of intersection are found, the vertices and half edges are labeled as in figure 6, by first traversing polygon 1 followed by polygon 2, numbering the half edges and vertices as they are encountered.

The next step is to determine the degree of each vertex. This is done by counting the number of occurrence of each vertex in the beginning and ending end point arrays. Opposite half edges are also introduced so one can proceed in either direction along the original line segments. The opposite half edge is simply a half edge directed in the opposite direction of the original half edges. Since, in figure 6 there are 33 line segments, the opposite half edges corresponding to the original half edges have indices 33 larger than the original half edges, as in figure 7. For example, half edge 31 goes from vertex 25 to vertex 12, while opposite half edge 64 goes from vertex 12 to 25. The half edges leaving a vertex must be associated with it in order of increasing angle of inclination. The angle of inclination is taken as zero parallel to the positive x-axis and increasing counter-clockwise. Consider, for example, vertex 3. The half edges leaving this vertex, in

Figure 6

Numbering of Half Edges and Vertices

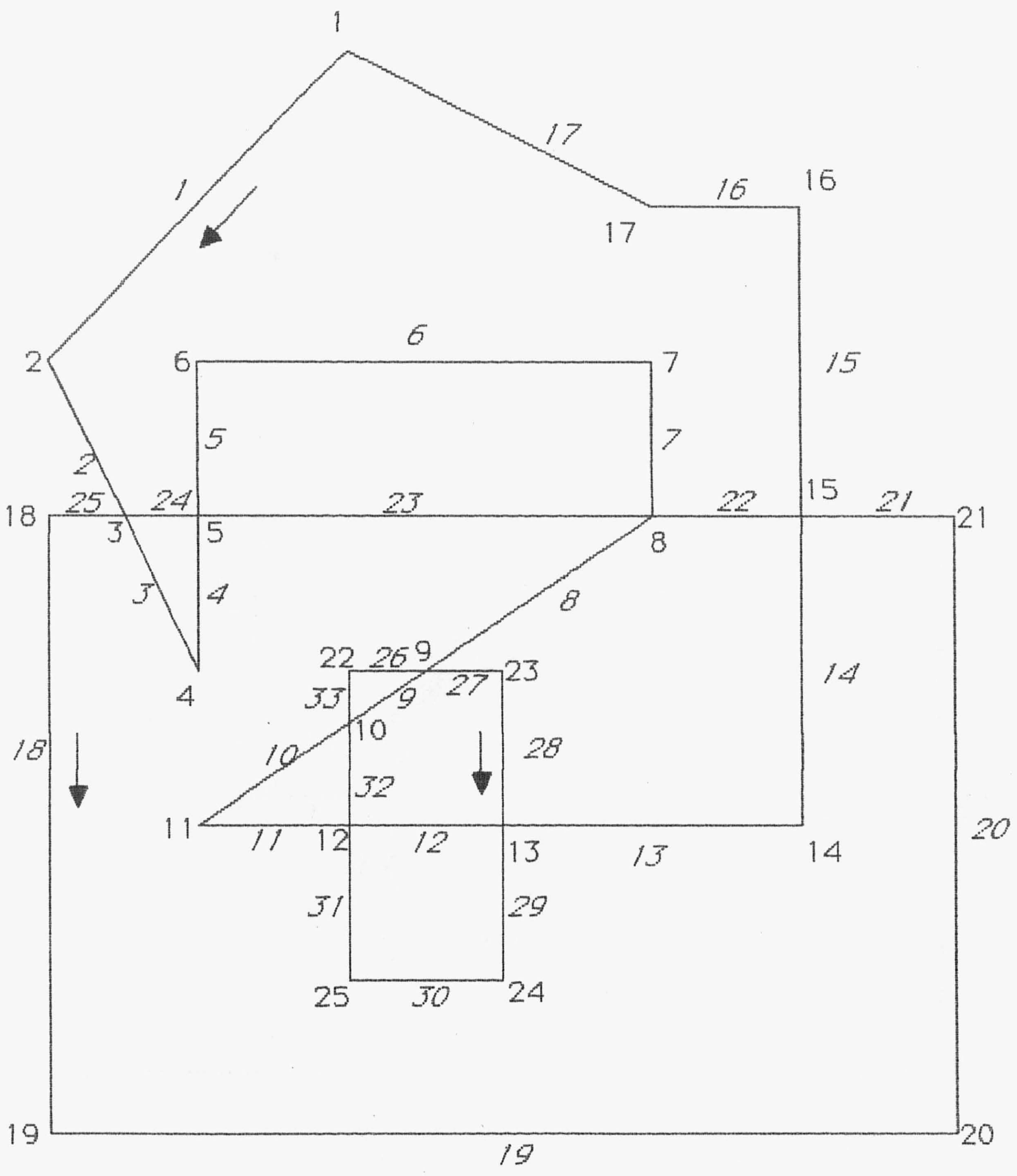
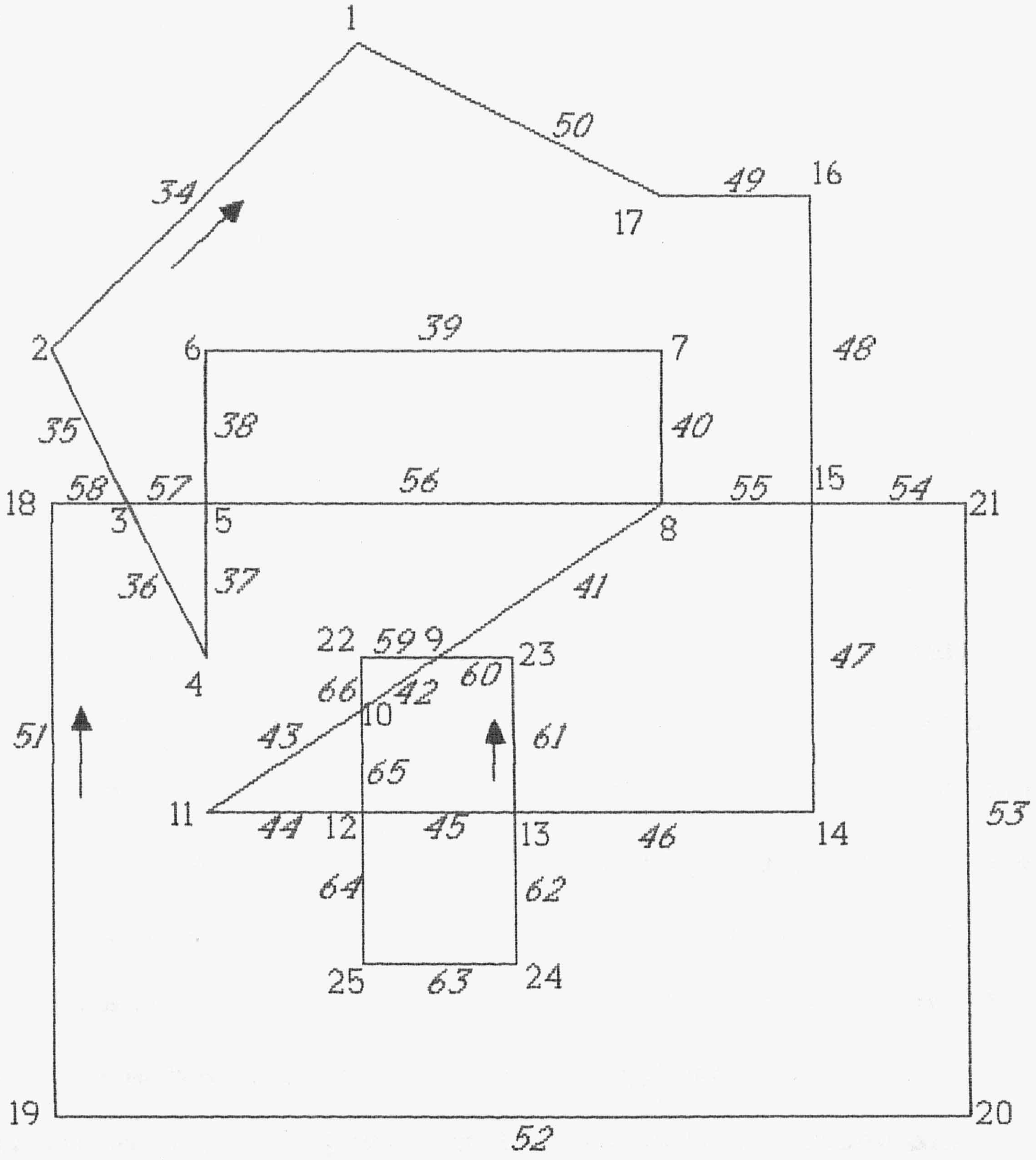


Figure 7

Numbering of Opposite Half Edges



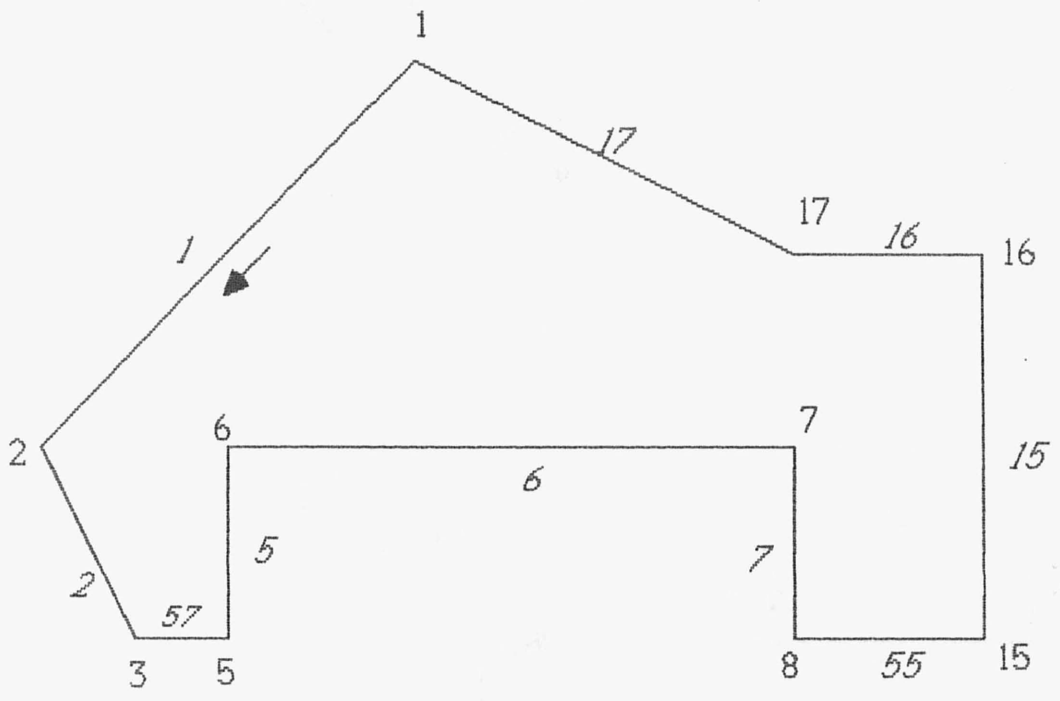
increasing order of the angle of inclination, are 57, 35, 25, and 3; with angles of 0, 116.6, 180, and 296.6 degrees respectively. When all of the vertices have been processed, the fundamental cycles of the graph are then computed.

The fundamental cycles are the boundaries of each of the components of the plane induced by the graph. They have the unique property that if a given point on the interior of the cycle is in the interior of a given polygon, then the entire interior of the cycle lies in that polygon. The first fundamental cycle is computed by selecting the unused half edge of lowest index, that is half edge 1. Half edge 1 is traversed from vertex 1 to vertex 2 and marked as used. The only half edge leaving vertex 2, except the trivial half edge 34, is half edged 2. Half edge 2 is marked as used and traversed to vertex 3, which is approached at an angle of 116.6 degrees. As stated earlier, the half edges leaving vertex 3 are sorted by increasing angle of inclination. The half edge of next smaller degree is selected as the edge to leave vertex 3 on. This is found to be half edge 57 with an angle of inclination of zero degrees. If the approach was made along the half edge of smallest angle, the half edge of largest angle is then selected to leave on. Half edge 57 is traversed from vertex 3 to 5 and is marked as used. This process is continued until cycle 1 is completed by traversing half edges 5, 6, 7, 55, 15, 16, and finally 17 as in figure 8. Cycle 2 is then computed in a similar manner. The unused half edge of lowest index is selected as the starting edge, this being half edge 3. This edge is traversed from vertex 3 to vertex 4, where edge 4 is then traversed to vertex 5. Finally edge 24 is traversed back to the starting vertex, thus completing cycle 2, as shown in the figure. At this point, half edges 1 through 7 have been used, hence the third cycle is begun by traversing line segment 8 producing cycle 3. The rest of the fundamental cycles are computed as shown in figure 9. The process stops after cycle 10, since all 66 of the half edges have been used in the cycles. Once the fundamental cycles are computed, it must be determined which polygon or polygons, if any, they are in. In order to determine if a particular cycle is part of the boundary of a particular polygon, a point must be chosen inside the cycle. Since the cycles are positively oriented with respect to the region they describe, this can be done by finding the normal vector, whose cross product with the first half edge of the cycle in question, is positive. Consider for example, cycle 3. The half edge of lowest index is half edge 8. The vector along half edge 8 is shown in figure 10a with coordinates <dx,dy>. Rotating this

Figure 8

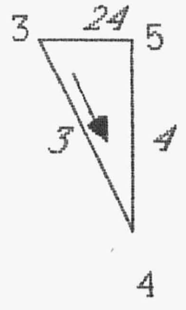
Building of the First Three Cycles

Cycle 1



Cycle 3

Cycle 2



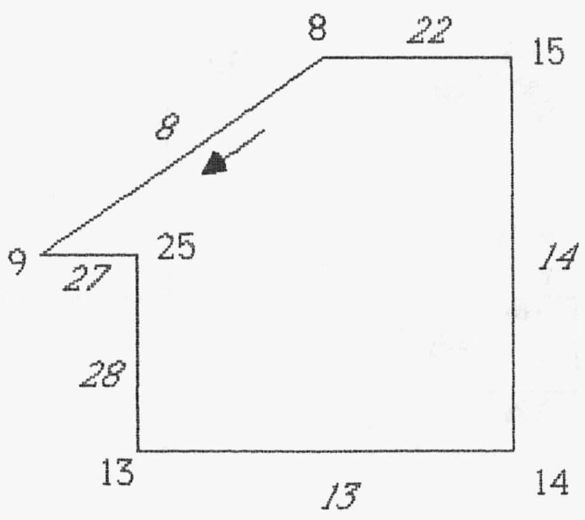


Figure 9

Remaining Cycles

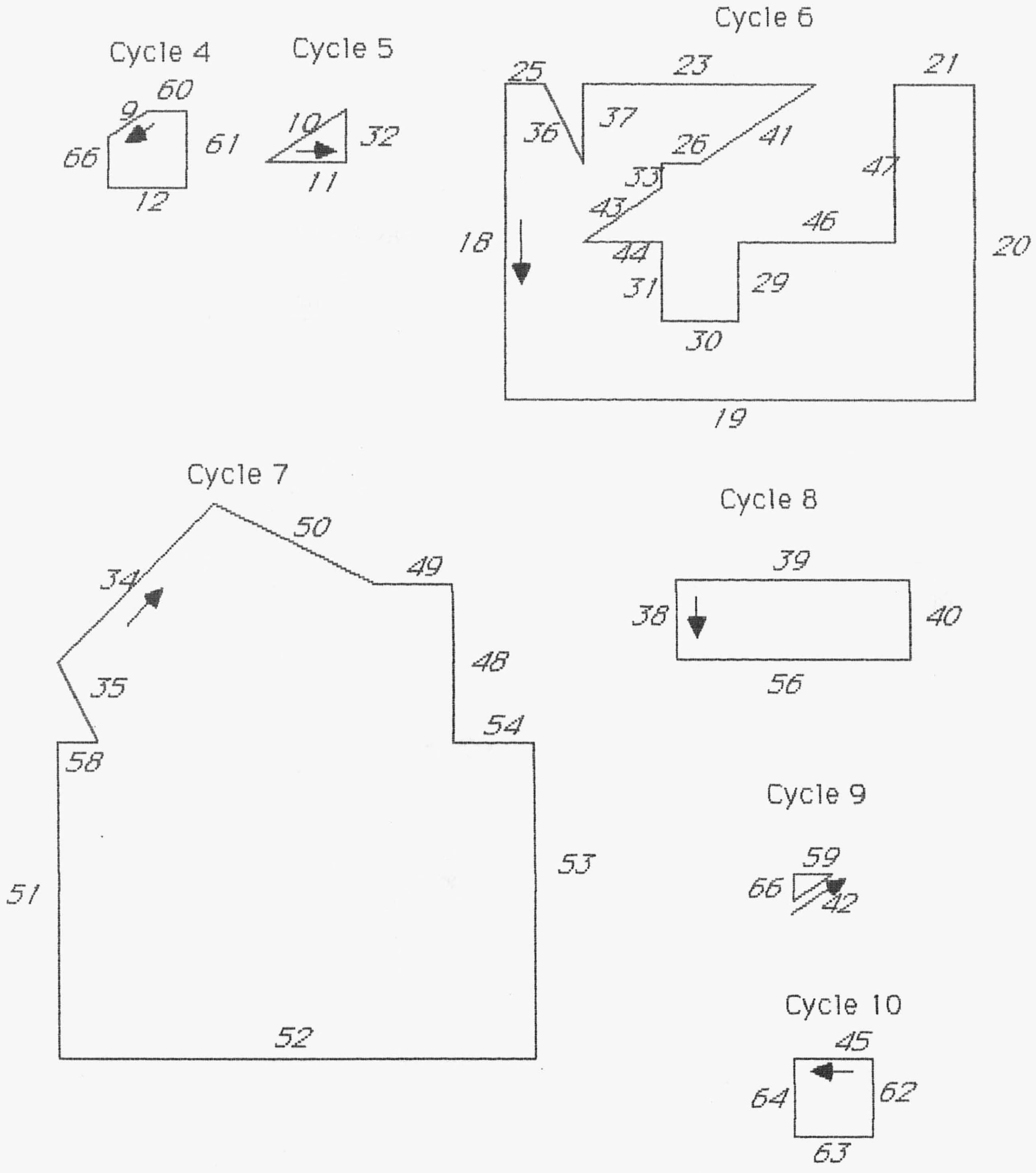
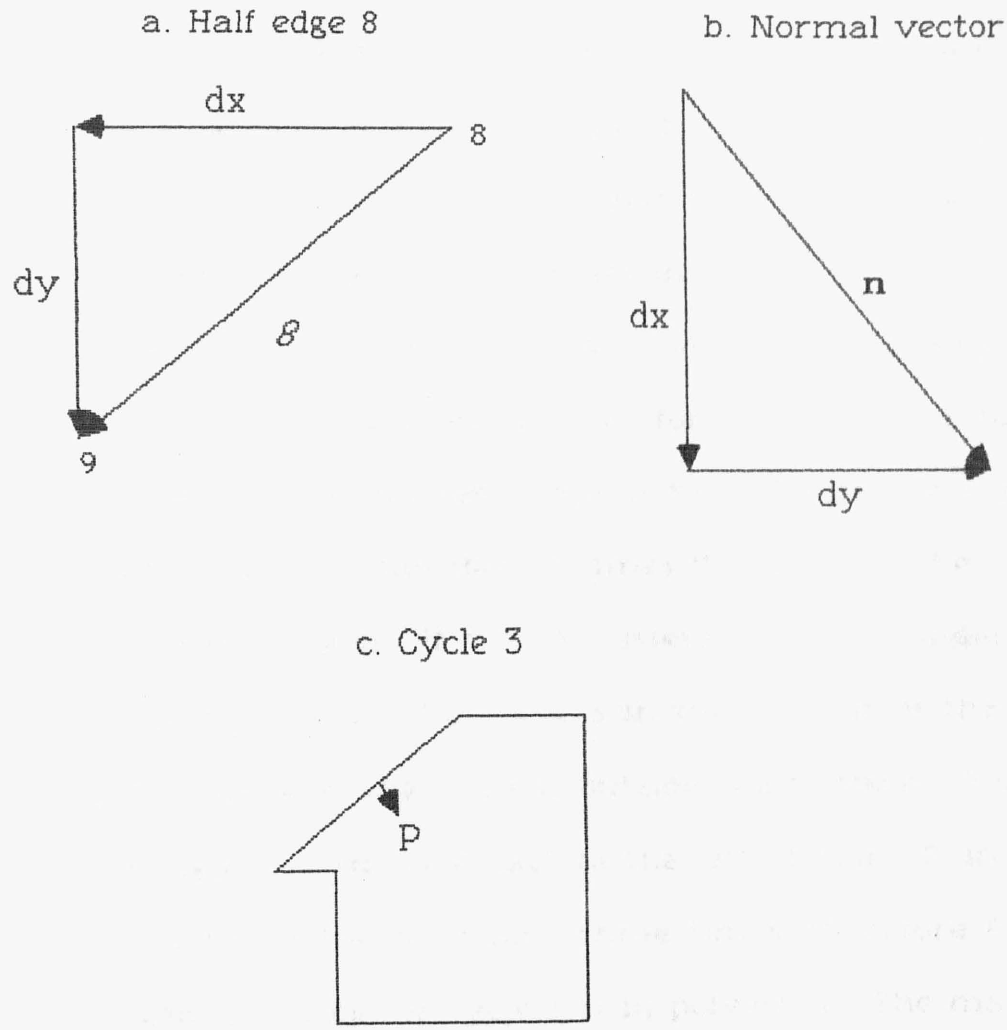


Figure 10

Construction of P in the Interior

of Cycle 3



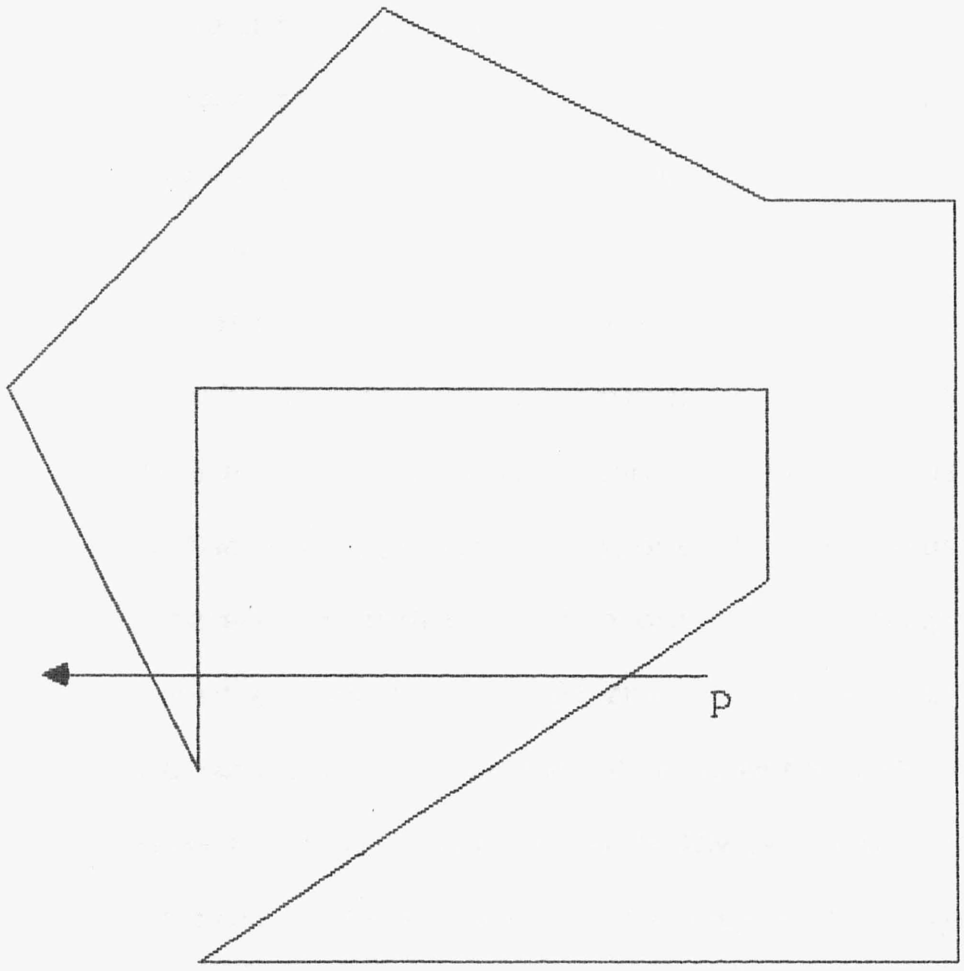
vector 90 degrees counter-clockwise causes it to appear as in figure 10b with coordinates <dy,-dx>. This vector is then made very short, so as not to extend beyond the other side of the cycle by first making it a unit vector and multiplying it by a very small number, epsilon. This vector is then added to the midpoint of the half edge, as in figure 10c. This new point, point P, lies within cycle 3 by the way it was constructed, so in order to tell which polygons this cycle is in, it must be determined which polygons point P lies in, if any.

It is obvious if the point lies in a polygon by observing the graph, however, it is more difficult for the computer to determine. One of the simplest ways is to cast a ray in some direction and count the number of times it intersects the boundary of the polygon. If the ray intersects the boundary an odd number of times, the point is in the interior of the of the polygon; otherwise the point is outside the polygon. For example, in figure 11, the ray cast to the left of point P, in polygon 1, intersects the boundary three times, therefore P is inside polygon 1, and hence cycle 3 is in polygon 1. The major difficulties with this method occur if the ray intersects a horizontal segment or if the ray intersects a vertex.

Intersecting a horizontal line segment does not need to be counted and will not affect the results. In the case of

Figure 11

Point P in the Interior of Polygon 1



intersecting a vertex, any line segment extending from this vertex with the other vertex below the intersected vertex will be counted, while all others will not be counted. For example, in figure 12, the ray from R intersects the boundary at vertex H, this is counted as intersecting line segment 7, but not 8, giving one intersection indicating R is inside the polygon. The ray from S intersects the boundary at vertices D and F and line segment 7. This is counted as intersecting line segments 3, 4, 5, 6, and 7 yielding 5 intersection, indicating S is inside the polygon. Finally consider the ray cast from point T. It intersect the boundary at vertex E and line segments 6 and 7. The intersection with vertex E is not counted since the line segments are above the ray. This leaves two intersection, indicating T is outside the polygon. Hence, by choosing a point on the interior of the cycle as described, and counting the number of intersections the cycles are classified as in Table I. Once each of the polygons can be described in terms of the fundamental cycles, it is simple to select the cycles that are required for a given operations, as shown in Table II.

Figure 12

Examples of the Difficulties with the

Cycle Classification Method

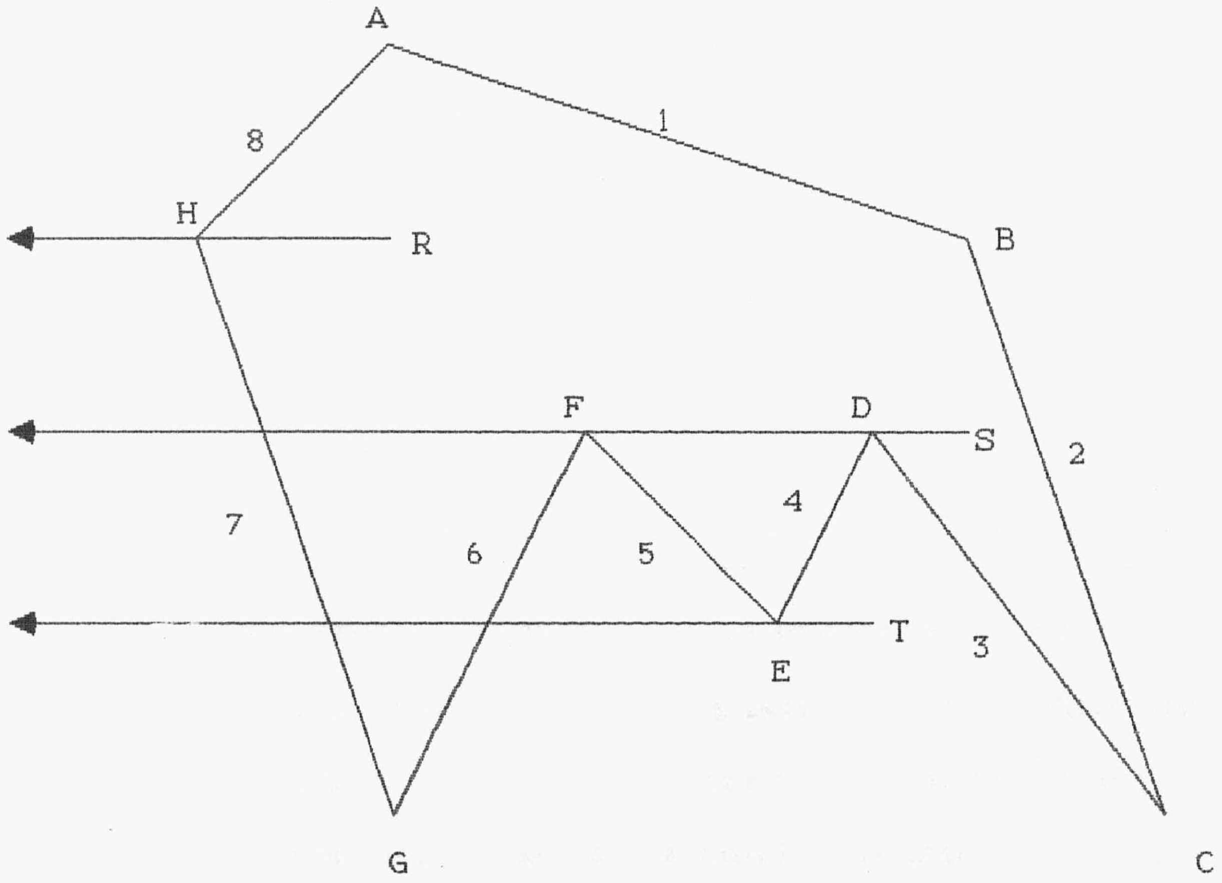


TABLE I

CLASSIFICATION OF CYCLES BY CONTENT

Cycle	Polygon 1	Polygon 2
1	XXX	
2	XXX	XXX
3	XXX	XXX
4	XXX	
5	XXX	XXX
6		XXX
7		
8		
9		
10		

The only problem that remains is to reconnect the desired cycles to create the resulting polygon. This is done by checking each cycle against the others to determine if they have a common edge. Take for example, the intersection of polygon 1 and the complement of polygon 2. Table II shows that this region is comprised of cycle 1 and cycle 4. Since these two cycles do not have a common edge, their graph shows up as two disjoint cycles, as in figure 13. A more complex example is the union of polygon 1 and 2.

Figure 13

Polygon 1 Intersect the Complement of Polygon 2

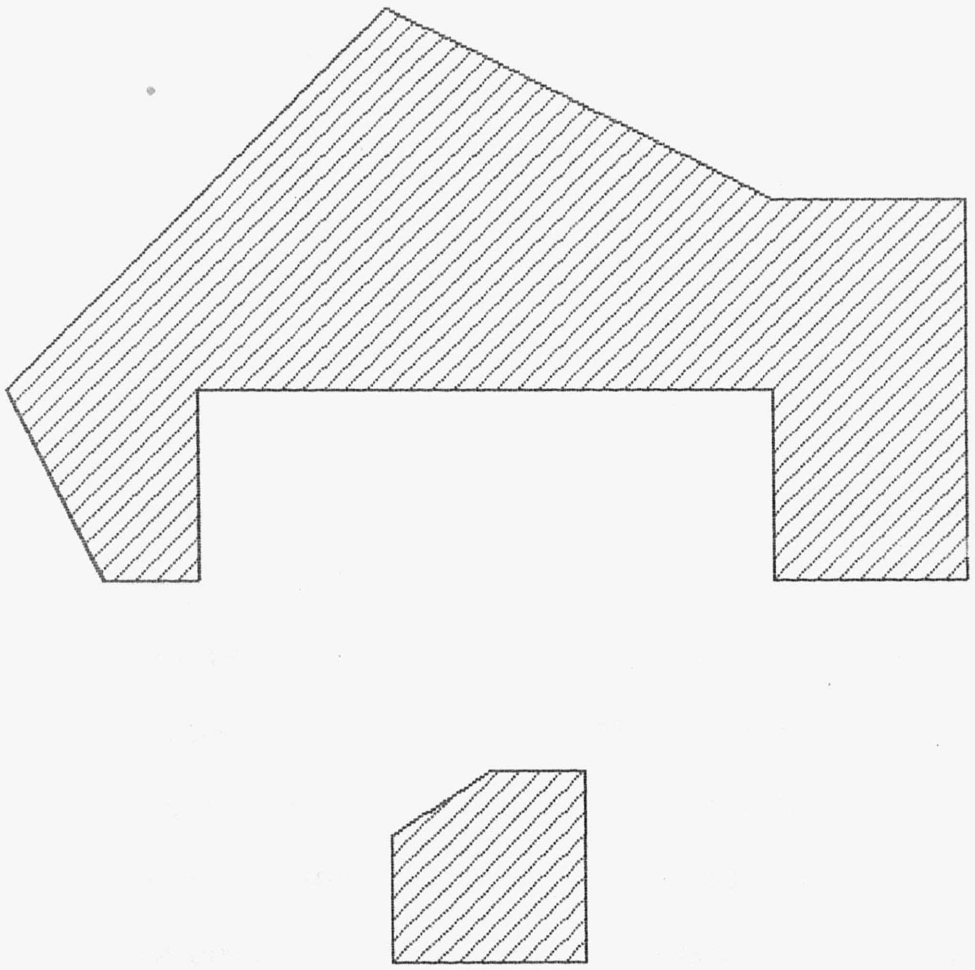


TABLE II

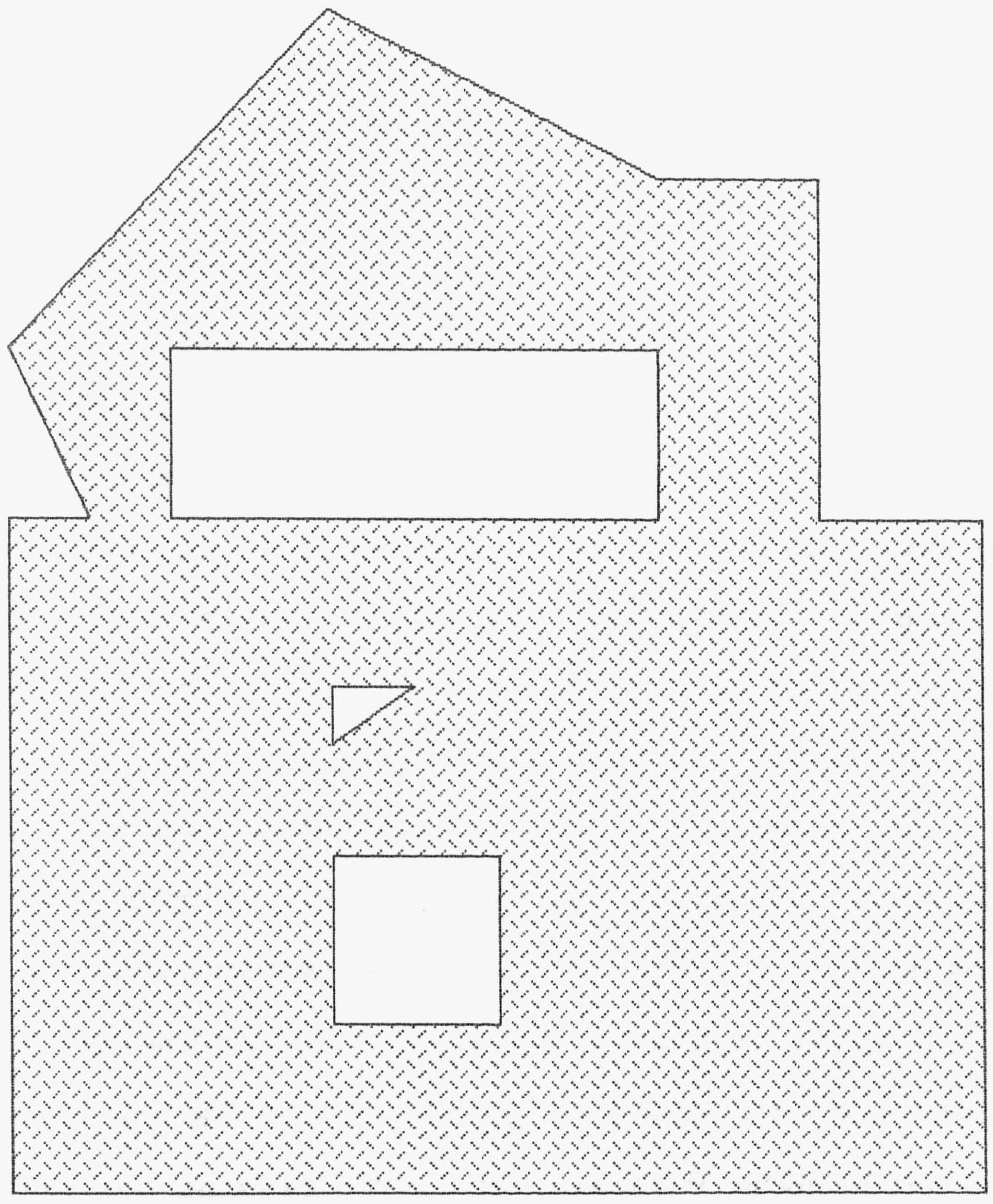
SELECTION RULES FOR BOOLEAN OPERATIONS

Cycle	$1 \cap 2$	$1 \cup 2$	$1 \cap 2'$	$1' \cap 2$	$1 \cup 2'$	$1' \cup 2$
1		XXX	XXX		XXX	
2	XXX	XXX			XXX	XXX
3	XXX	XXX			XXX	XXX
4		XXX	XXX		XXX	
5	XXX	XXX			XXX	XXX
6		XXX		XXX		XXX
7					XXX	XXX
8					XXX	XXX
9					XXX	XXX
10					XXX	XXX

This region is made up of the first 6 cycles, however many of the cycles have common edges. These common edges are easy to detect since the indices differ by 33, the number of line segments. Cycle 1 and cycle 2 have the common edge pair 24 and 57, which connect vertices 3 and 5. In order to join these two cycles, the boundary of cycle 1 is traced to the first occurrence of either of these vertices. Vertex 3 is the first one encountered, then cycle 2 is traced to vertex 5 and then cycle 1 is followed. This process is continued until the region in figure 14 is produced. This completes the discussion of the two-dimensional case.

Figure 14

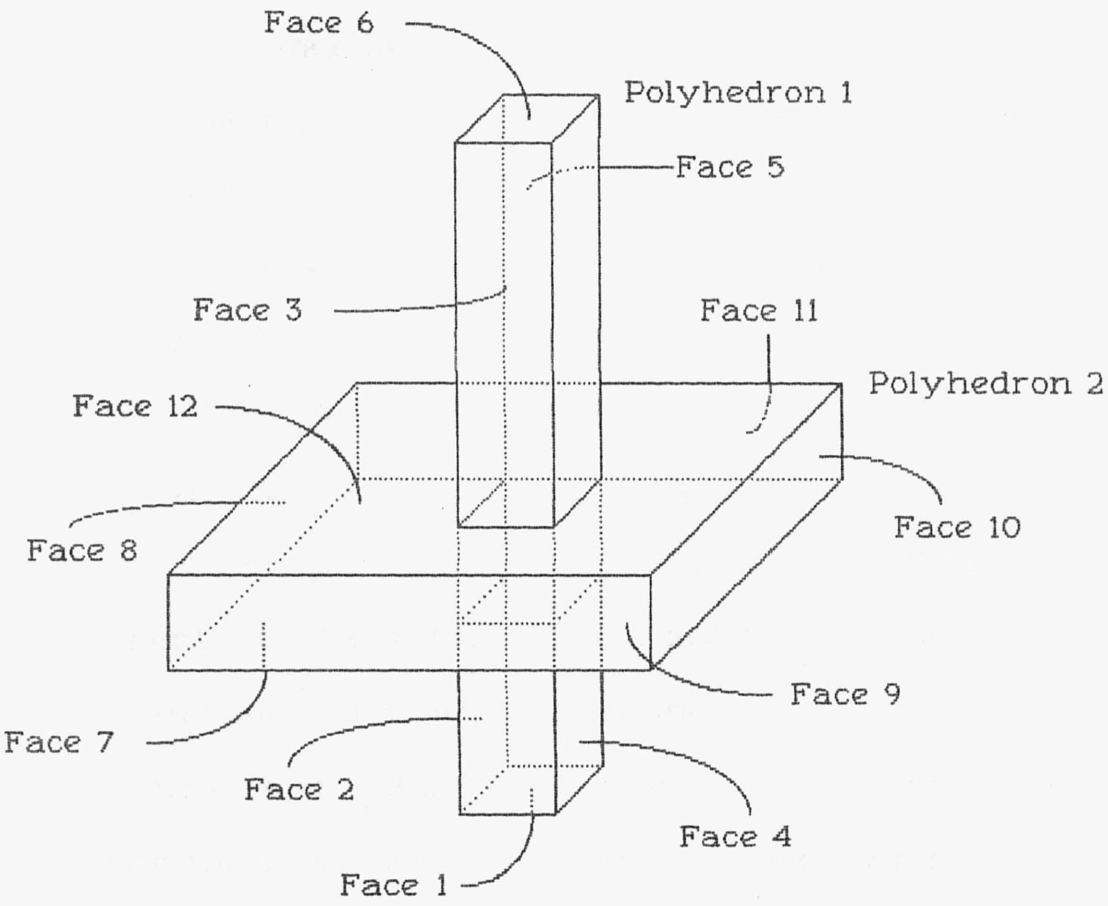
Polygon 1 Union Polygon 2



In order to operate on three-dimensional polyhedrons, it is possible to reduce them into numerous two-dimensional polygons describing the faces of the polyhedrons. Each of the polygonal faces must be broken up into facets. The facets of each face are determined by intersecting each face of the polyhedron with each of the faces of the other polyhedron. These facets are the boundaries of each of the components of three-space induced by the polyhedrons. Consider, for example, the polyhedrons in figure 15. Each of the polyhedrons have six faces. They are numbered as in the figure. The intersection of the faces is found by first parameterizing each of the face planes with $(\mathbf{n}|\mathbf{x})=\mathbf{b}$, where $\mathbf{n}$ is the normal vector, $\mathbf{x}$ is any point in the plane, and $\mathbf{b}$ is determined by the plane. The planes are then checked to see if they coincide or are parallel, by checking if the normal vectors are linearly dependent. If the planes coincide, the polygonal faces are intersected as described earlier. Whereas, if it is determined that the planes intersect, the point on the line of intersection closest the origin is found by using Lagrange multipliers to minimize the length of $\mathbf{x}$ subject to $(\mathbf{n_1}|\mathbf{x})=\mathbf{b_1}$ and $(\mathbf{n_2}|\mathbf{x})=\mathbf{b_2}$. With the parameterization of the line, it is possible to detect if the line of intersection cuts through the faces, indicating the faces intersect. This is the method

Figure 15

Polyhedrons 1 and 2



used to find the cross edges and self edges of each of the faces.

The cross edges are the new edges created by the intersection of this face with another face, while the self edges are subdivisions of the original edges caused by the intersection with other faces. The new edges are linked to the original polygonal face creating a new face composed of facets. The boundaries of the facets are the fundamental cycles of this new polygon that are part of the original face. Consider, for example, face 2 in figure 16. The original face is cut by face 7 of polyhedron 2, creating cross edge 1; while the intersection with face 12 creates cross edge 2. These new cross edges divide two of the original edges into six new self edges. The fundamental cycles of this polygon are found as described previously, and those that are part of the original face are kept as the new facets. Face 2 is composed of three facets as shown. Once the faces are broken up into facets, the facets required to describe the result of a given Boolean operation must be selected. This is done by taking a point in the facet and going a small distance along the normal, to obtain a point on the outside of the facet, and going a small distance in the direction opposite the normal to pick a point on the inside of the facet. These points are then evaluated to see which solid or solids, if any, they are in. In order to determine which

Figure 16

Numbering of Facets in Polyhedron 1

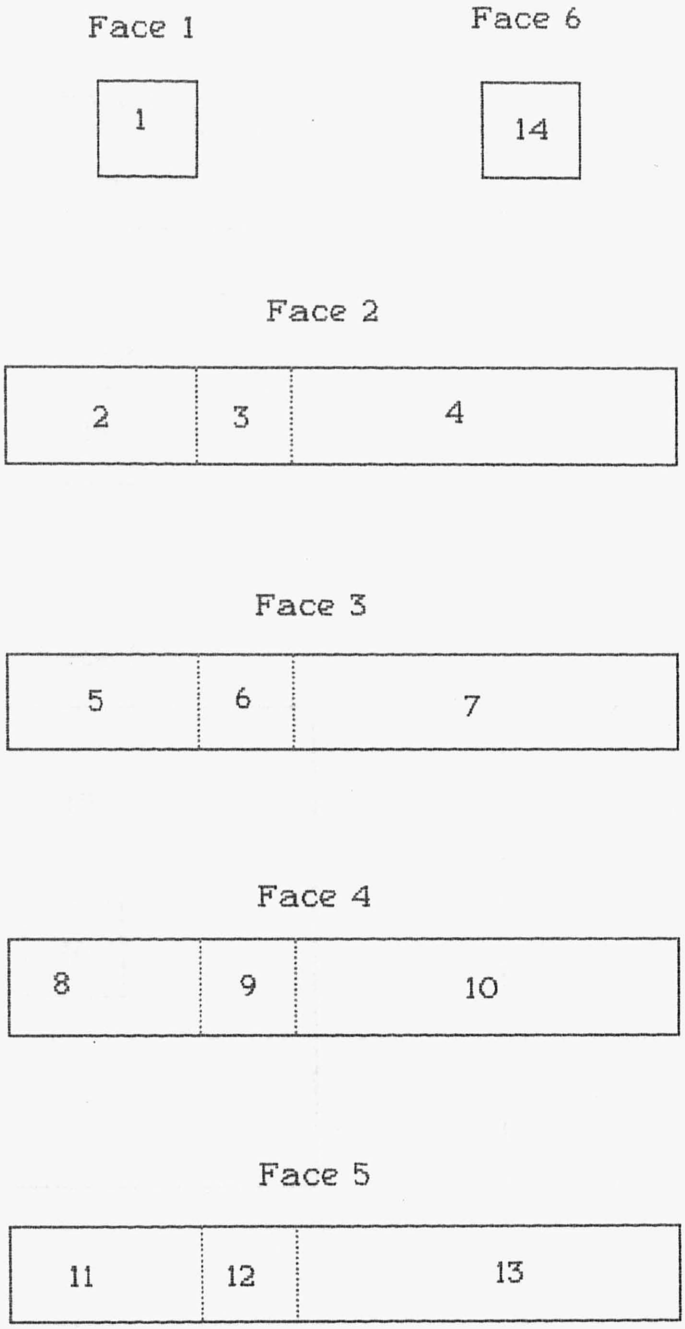
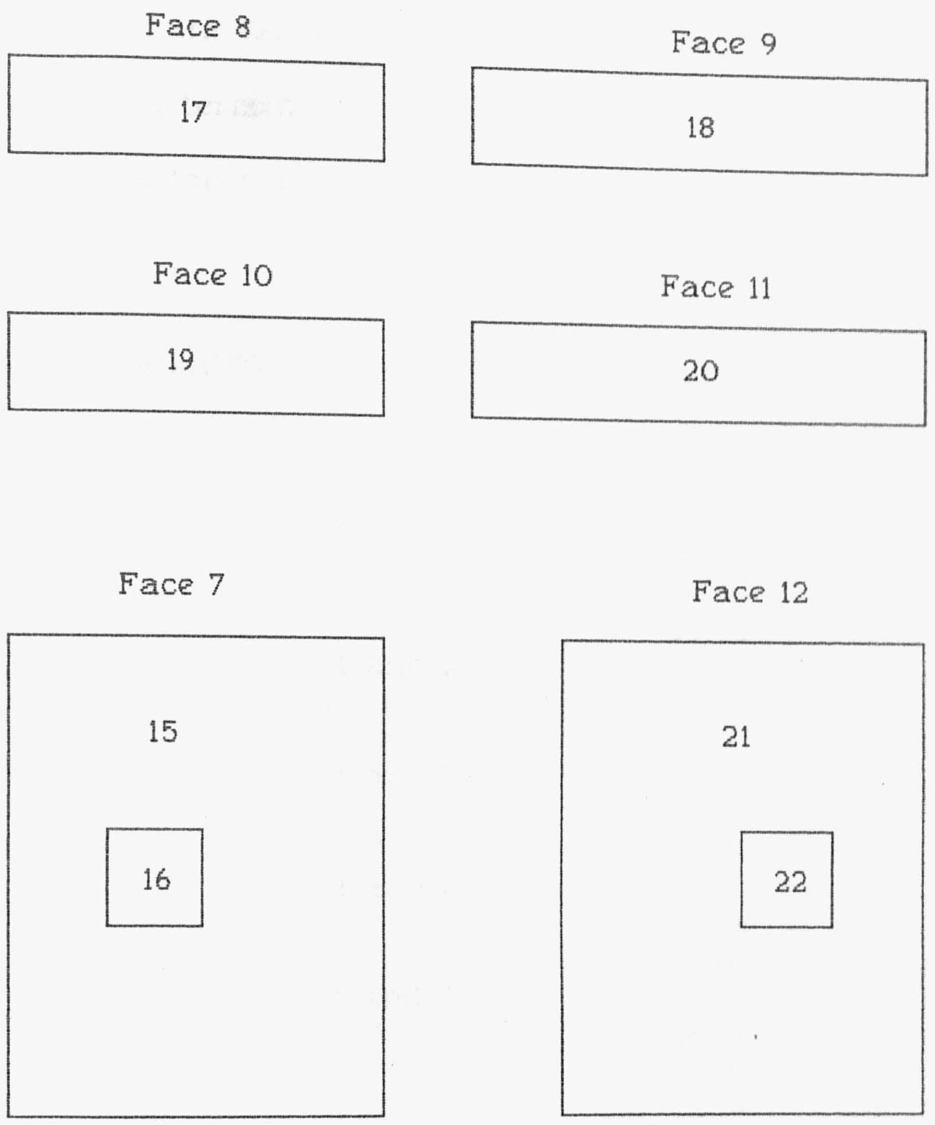


Figure 17

Numbering of Facets in Polyhedron 2



solid a point is in, the distance from the point to all of the faces of the solid is computed, and if the distance to the closest face is positive, then the point is in the interior of the solid since all of the faces have outward normals. Hence the facets are characterized according to which component lies on each side of the facet as in Table III.

TABLE III

CHARACTERIZATION OF FACETS BY ADJOINING REGIONS

Facet	Inside	Outside
1	1	none
2	1	none
3	1 and 2	2
4	1	none
5	1	none
6	1 and 2	2
7	1	none
8	1	none
9	1 and 2	2
10	1	none
11	1	none
12	1 and 2	2
13	1	none
14	1	none
15	2	none
16	1 and 2	1
17	none	2
18	none	2
19	none	2
20	none	2
21	2	none
22	1 and 2	1

It is now possible to select the required facets to generate the polyhedron resulting from a given Boolean operation. Suppose the intersection of the two polyhedrons is requested. Any facet with both polyhedrons on one side are selected, they are 3, 5, 9, 12, 16, and 22. These facets are all oriented correctly since the desired region is all on the inside, the side opposite the outward normal. For a more complicated example, consider the intersection of polyhedron 2 with the complement of polyhedron 1. The facets should have only polyhedron 2 on one side. The selected facets are 2, 6, 9, 12, 15, 17, 18, 19, 20, and 21. However, all of the facets except 15 and 21 are oriented incorrectly since polyhedron 2 lies on the outside of these facets. In order to correct the orientation, since the polyhedron will be expressed in terms of outward normals, the order of the traversal of the vertices in these facets are reversed. The result of any Boolean operation can be found in a similar manner.

Conclusion

The methods described in this paper have been implemented in the development of a computer program in FORTRAN which performs Boolean operations on polygons and polyhedrons. Dr. McGrath and I have spent numerous hours designing and testing the program which confirms the validity of this method. Although there are many different methods described in current literature, this method is a combination of several of them and is very conducive to computer graphics. Another advantage is that the method is relatively straight forward and can easily be modified for any particular application.

BIBLIOGRAPHY

Foley, James D. and Andries Van Dam. _Fundamentals of Interactive Computer Graphics_. Reading, MA: Addison-Wesley Publishing Co., 1982.

Franklin, Wm. Randolph, "Efficient Polyhedron Intersection and Union." _Graphics Interface_, 1982, 73-80.

Newman, William M. and Robert F. Sproull. _Principles of Interactive Computer Graphics_. New York: McGraw-Hill Book Co., 1979.

Pavlidis, Theo. _Algotithms for Graphics and Image Processing_. Rockville, MD: Computer Science Press, 1982.

Preparata, Franco P. and Michael Ian Shamos. _Computational Geometry_. New York: Springer-Verlag, 1985.

Requicha, Aristides A. G. and Herbert B. Voelcker. "Boolean Operations in Solid Modelling: Boundary Evaluation and Merging Algorithms." _Technical Memorandum Number 26_. Production Automation Project. University of Rochester, October 1984.

Tilove, Robert B. "Exploiting Spatial and Structural Locality in Geometric Moddeling." _Technical Memorandum Number 38_. Production Automation Project. University of Rochester, October 1981.